Лабораторна робота №2

Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

КВ-94 Суховейко Олексій

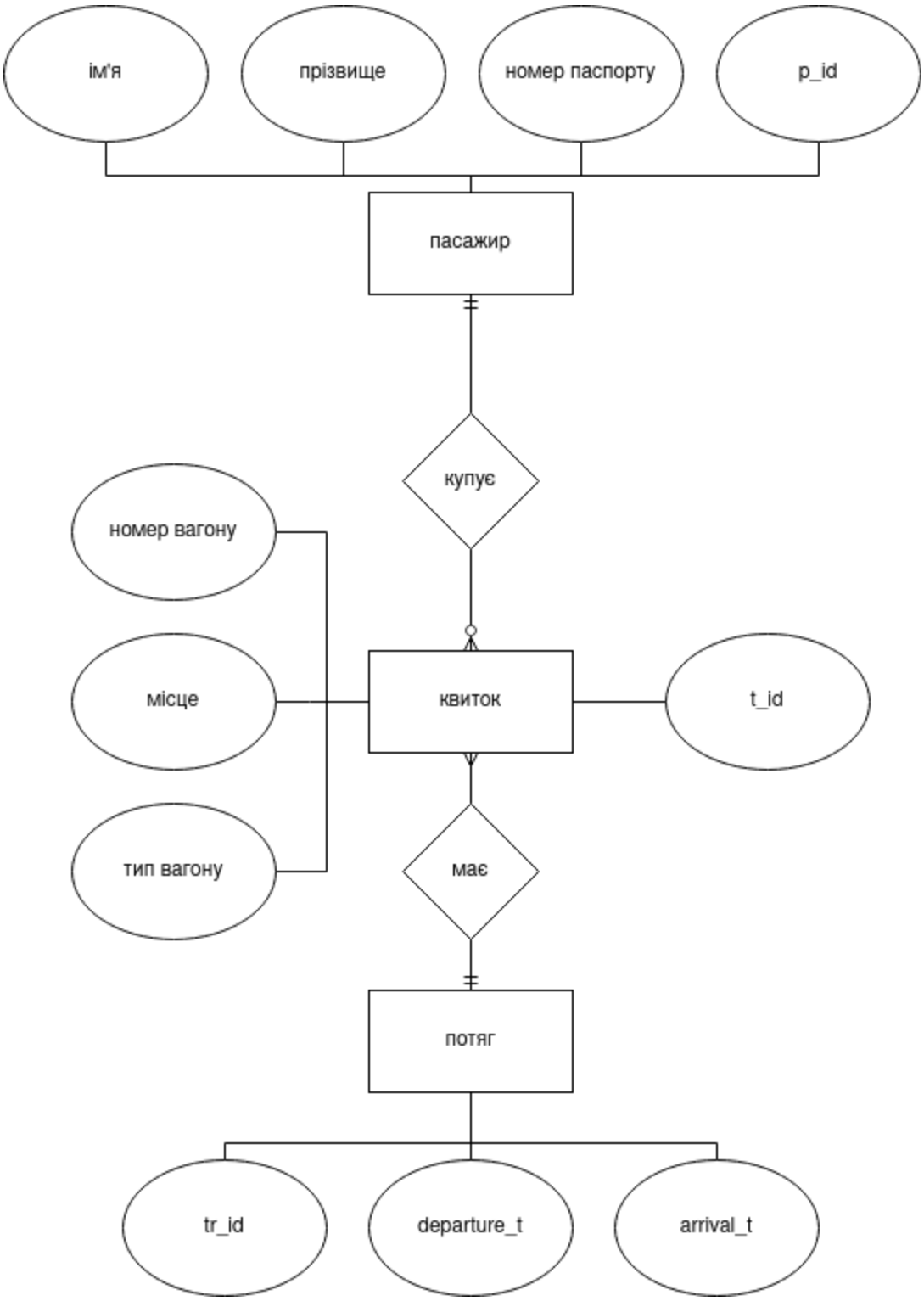
Завдання

Загальне завдання роботи полягає у наступному:
Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

URL репозиторію з вихідним кодом

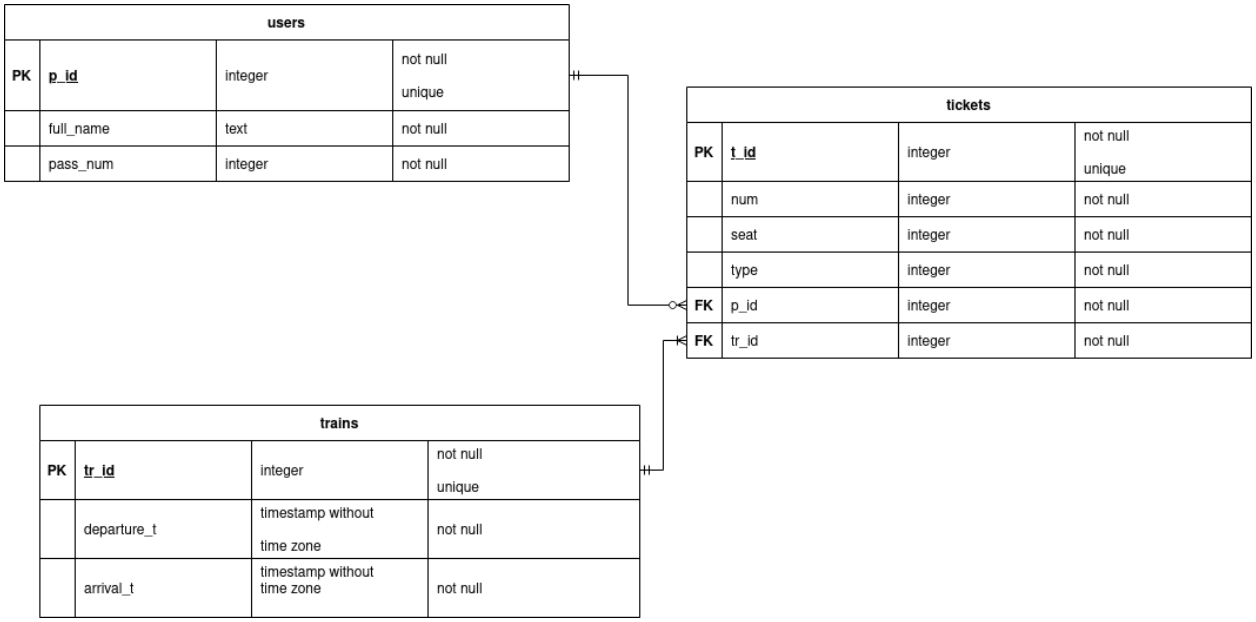
https://github.com/meltedit/db-labs/tree/master/lab2

Модель та структура бази даних



Модель “сутність-зв’язок” для обраної предметної області

-



Опис структури БД

Таблиця	Атрмбут	Опис атрибуту	Тип	Обмеження
users	p_id	Унікальний ідентифікатор	integer	not null unique
	full_name	Повне ім’я користувача	text	not null
	pass_num	Номер паспорту	integer	not null
trains	tr_id	Унікальний ідентифікатор	integer	not null unique
	departure_t	Час віправлення	timestamp without time zone	not null
	arrival_t	Час прибуття	timestamp without time zone	not null
tickets	t_id	Унікальний ідентифікатор	integer	not null unique
	num	Номер вагону	integer	not null
	seat	Номер місця	integer	not null

	type	Тип вагону	text	not null
	p_id	Посилання на користувача	integer	not null
	tr_id	Почилання на потяг	integer	not null

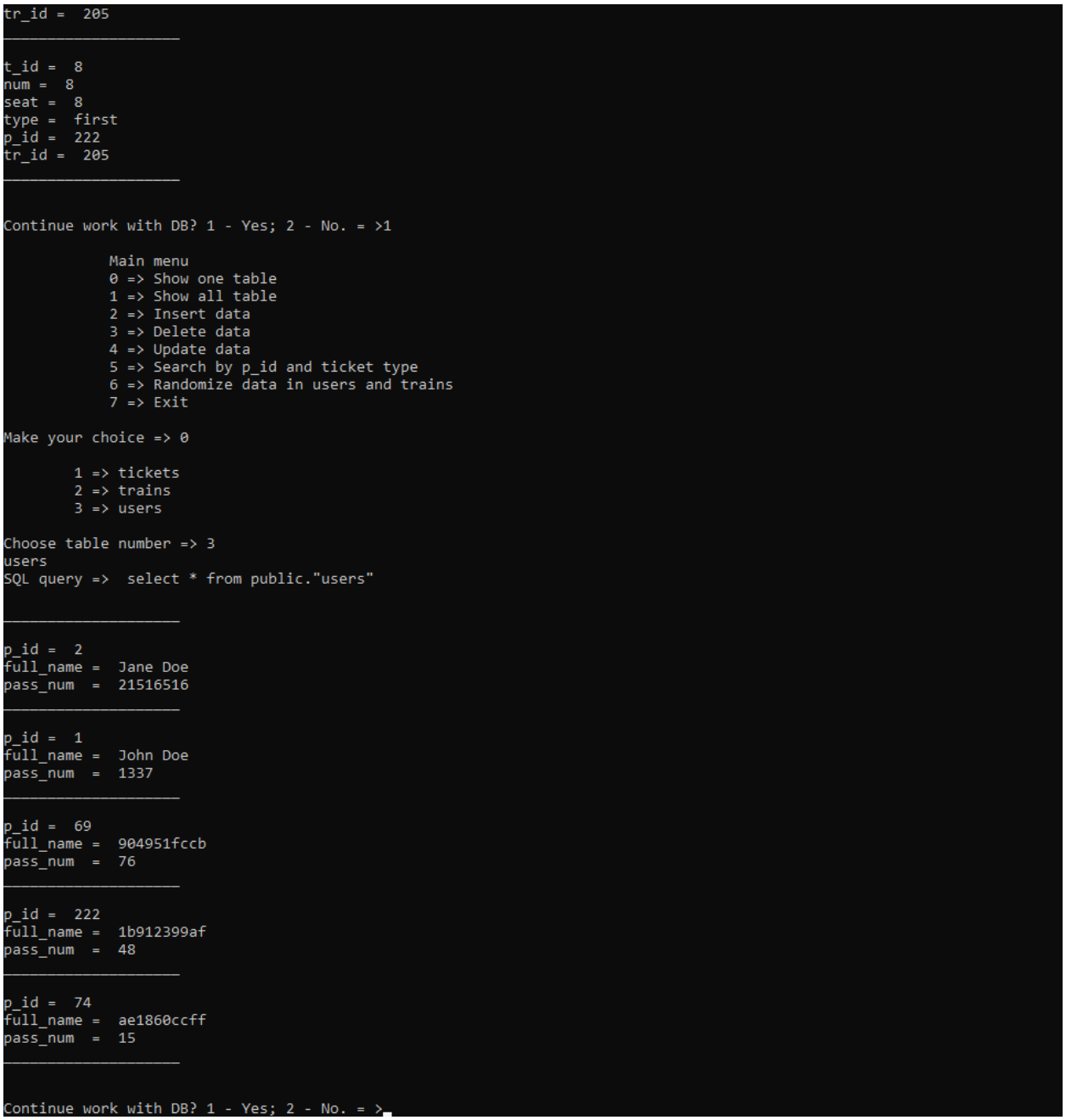
Мова програмування та бібліотеки

Робота виконана мовою програмування Python, для підключення до серверу Postgresql використано модуль psycopg2

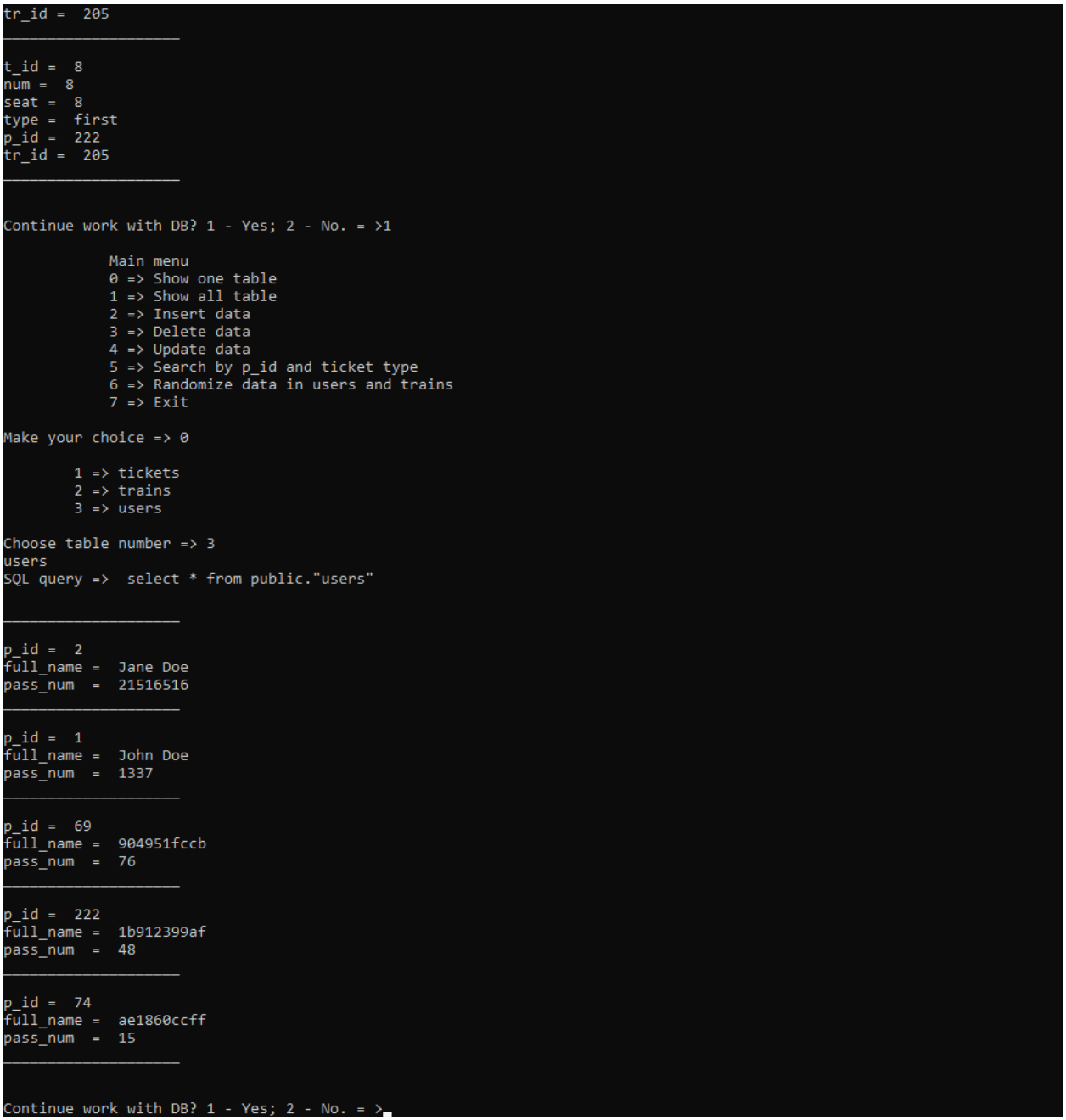
Пункт 1

Вилучення

Виведення батьківської таблці



Виведення дочірньої таблці



Вилучення запису батьківської таблиці

```
           6 => Randomize data in users and trains
           7 => Exit

Make your choice => 3

        1 => tickets
        2 => trains
        3 => users

Choose table number => 3
Delete user with ID = 74
users
SQL query =>  delete from "users" where "p_id"= 74
Data deleted successfully!

Continue deletion? 1 - Yes; 2 - No =>2

Continue work with DB? 1 - Yes; 2 - No. = >1

           Main menu
           0 => Show one table
           1 => Show all table
           2 => Insert data
           3 => Delete data
           4 => Update data
           5 => Search by p_id and ticket type
           6 => Randomize data in users and trains
           7 => Exit

Make your choice => 0

        1 => tickets
        2 => trains
        3 => users

Choose table number => 3
users
SQL query =>  select * from public."users"

____________________

p_id =  2
full_name =  Jane Doe
pass_num  =  21516516
____________________

p_id =  1
full_name =  John Doe
pass_num  =  1337
____________________

p_id =  69
full_name =  904951fccb
pass_num  =  76
____________________

p_id =  222
full_name =  1b912399af
pass_num  =  48
____________________


Continue work with DB? 1 - Yes; 2 - No. = >
```

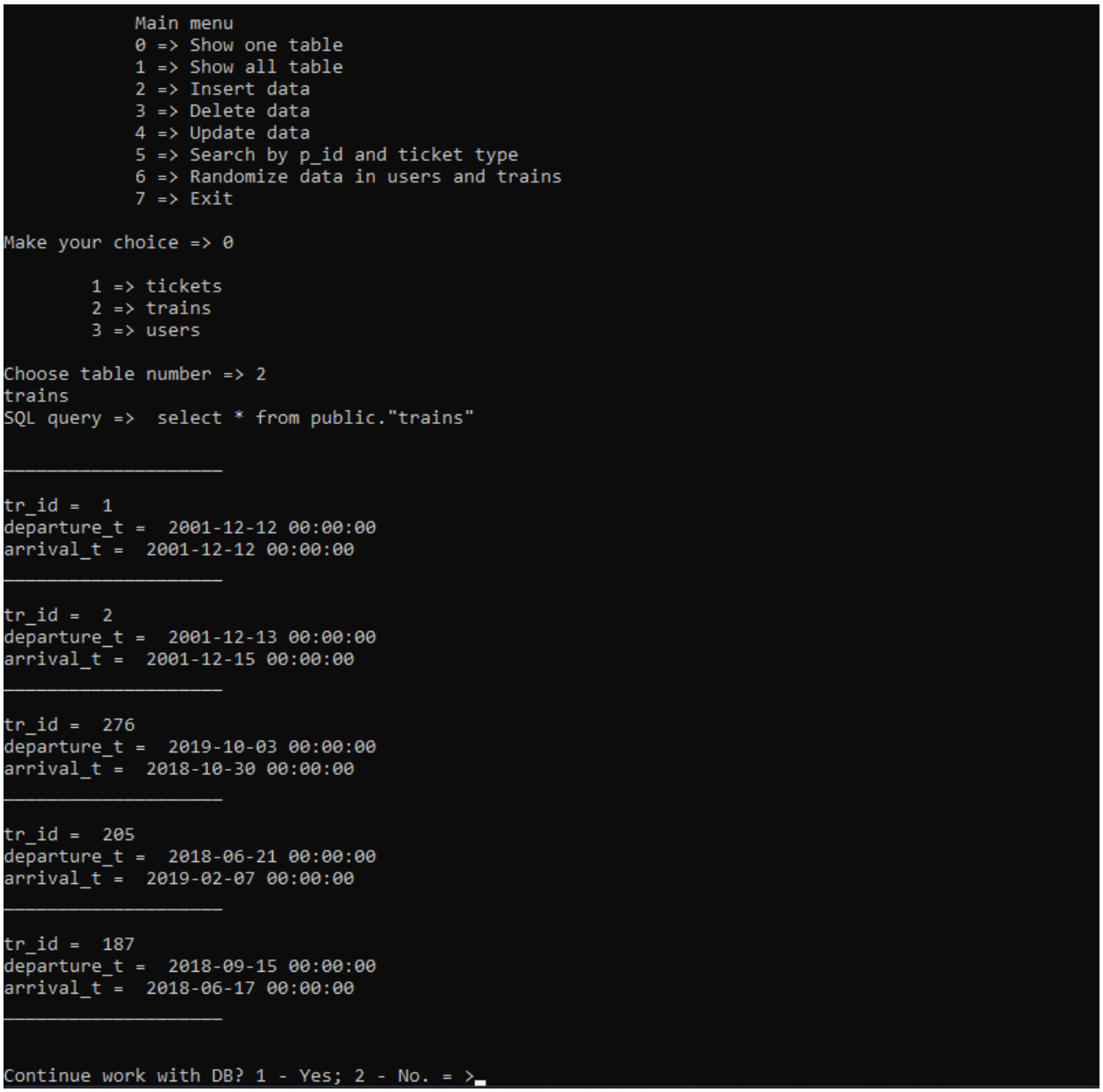
Вилучення залежних даних з батьківської таблиці

```
Continue work with DB? 1 - Yes; 2 - No. = >1

           Main menu
           0 => Show one table
           1 => Show all table
           2 => Insert data
           3 => Delete data
           4 => Update data
           5 => Search by p_id and ticket type
           6 => Randomize data in users and trains
           7 => Exit

Make your choice => 3

       1 => tickets
       2 => trains
       3 => users

Choose table number => 3
Delete user with ID = 222
users
SQL query =>  delete from "users" where "p_id"= 222
PostgreSQL Error:  update or delete on table "users" violates foreign key constraint "tickets_p_id_fkey" on table "ticke
ts"
DETAIL:  Key (p_id)=(222) is still referenced from table "tickets".

Po0stgreSQL connection is closed
Press any key to continue . . .
```

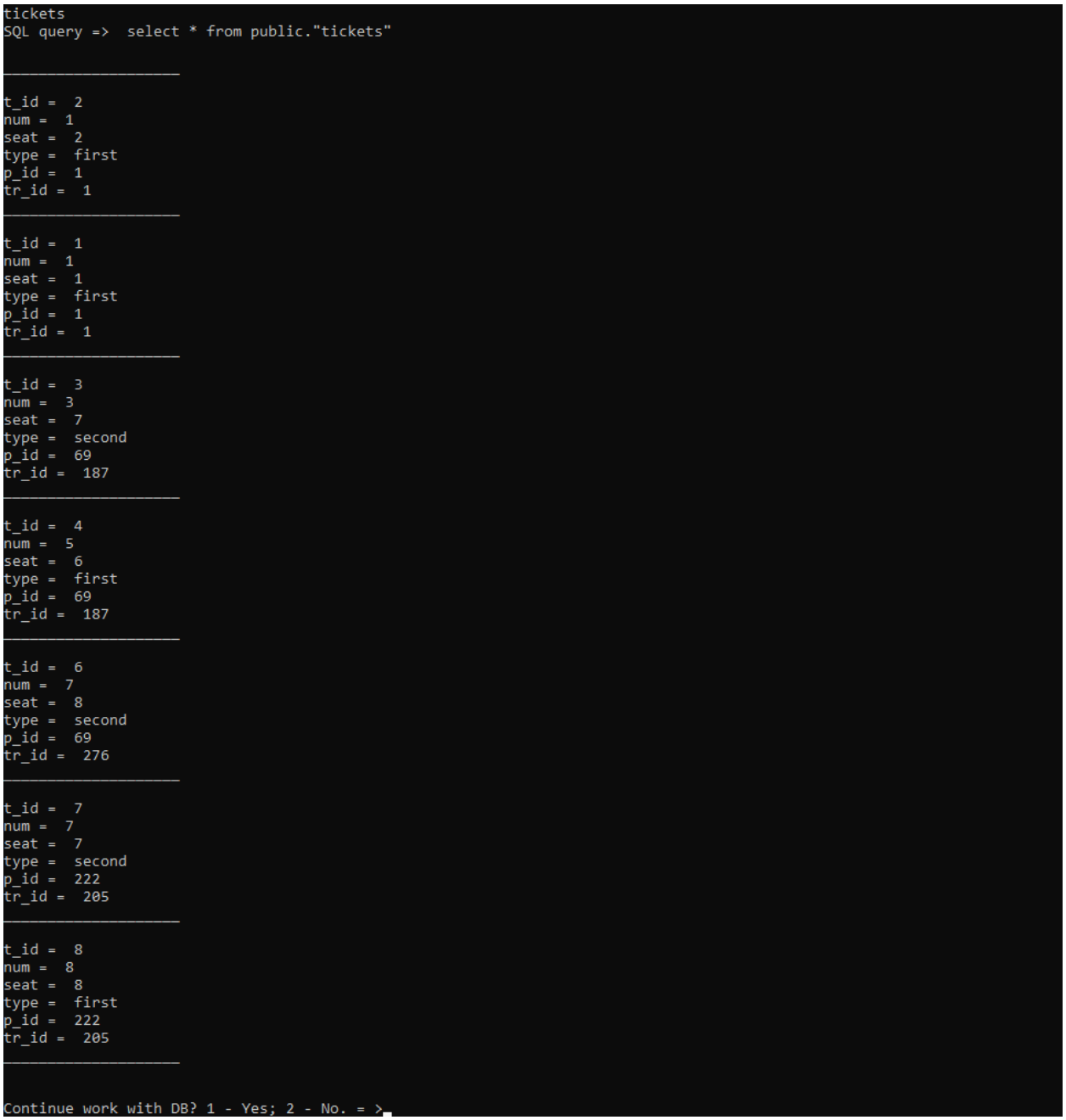
Перехоплена помилка від сервера

Вставка

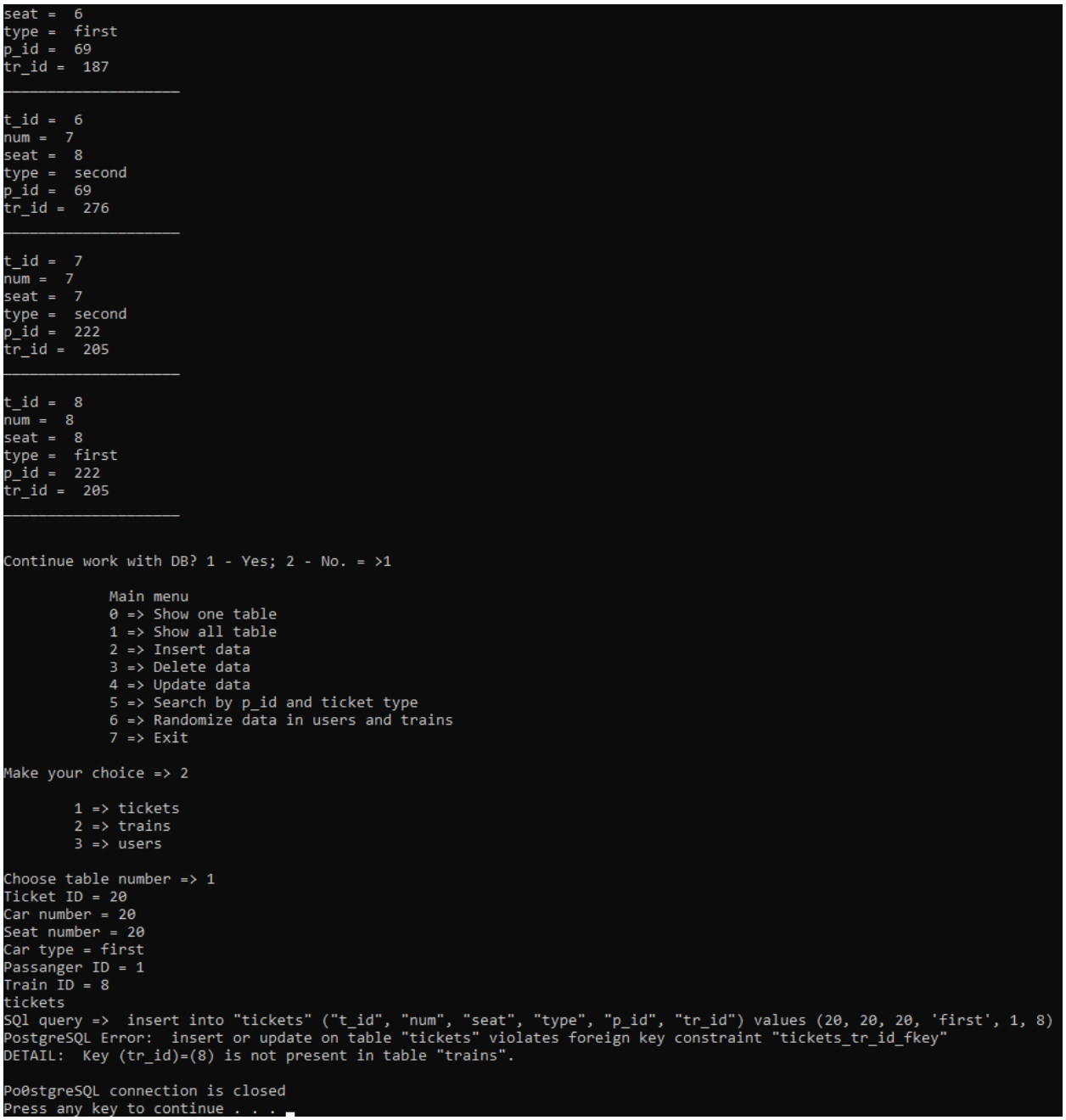
Виведення батьківської таблиці



Виведення дочірньої таблиці

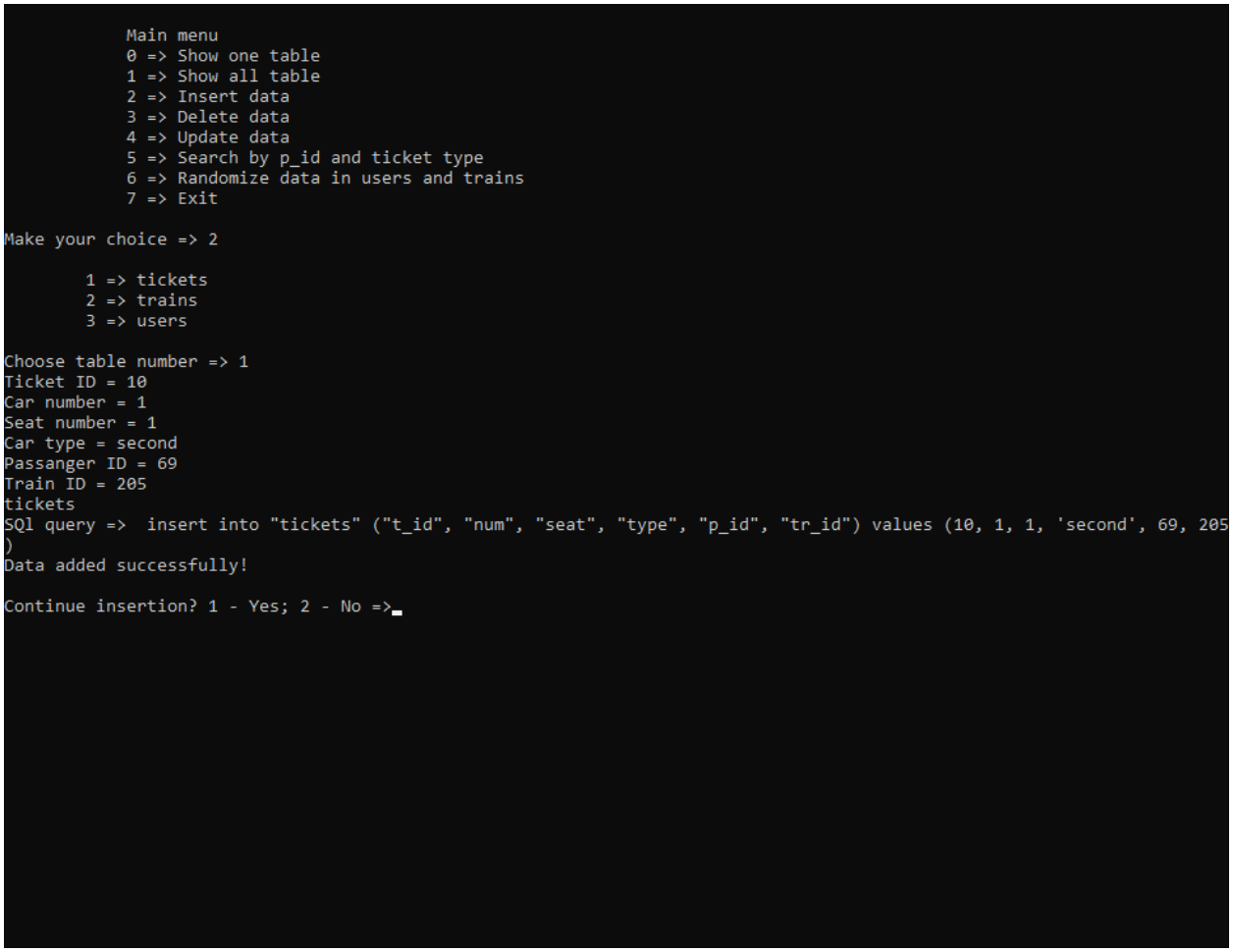


Вставка запису в дочірню таблицю, неіснуючого запису в батьківскій



Перехоплена помилка від сервера

Вставка існуючого запису в дочірню таблицю



Вміст таблиці після вставки

	t_id [PK] integer	num integer	seat integer	type text	p_id integer	tr_id integer
1	1	1	1	first	1	1
2	2	1	2	first	1	1
3	3	3	7	second	69	187
4	4	5	6	first	69	187
5	6	7	8	second	69	276
6	7	7	7	second	222	205
7	8	8	8	first	222	205
8	10	1	1	second	69	205

Пункт 2

Введення 10 рандомізованих записів у таблиці users та trains

```
           Main menu
           0 => Show one table
           1 => Show all table
           2 => Insert data
           3 => Delete data
           4 => Update data
           5 => Search by p_id and ticket type
           6 => Randomize data in users and trains
           7 => Exit

Make your choice => 6
How many users to random? => 10
SQL query =>
        insert into "users" (p_id, full_name, pass_num)
        select (300*random())::integer+4,

        substr(md5(random()::text), 1, 10),
        (random() * 70 + 10)::integer
        FROM generate_series(1, 10);


SQL query =>
        insert into "trains" (tr_id, departure_t, arrival_t)
        select (300*random())::integer+4,

        DATE '2018-01-01' + (random() * 700)::integer,


        DATE '2018-01-01' + (random() * 700)::integer
        FROM generate_series(1, 10);


Data randomed successfully!

Continue randomizition? 1 - Yes; 2 - No =>
```

Вміст таблиць із згенерованними даними

trains

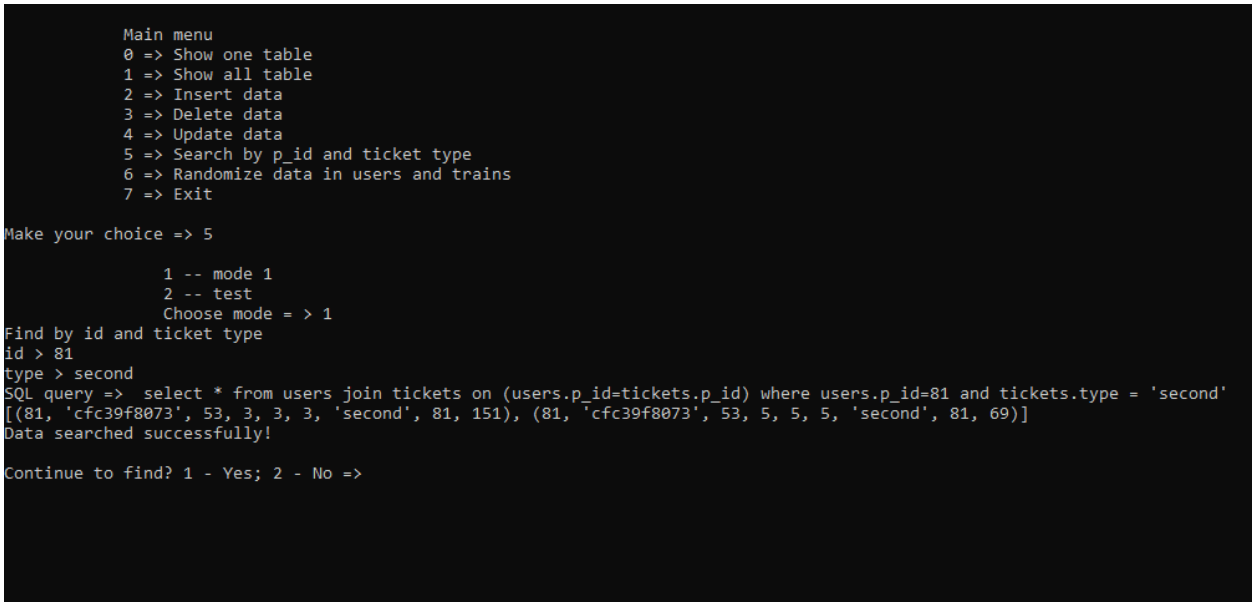
	tr_id [PK] integer	departure_t timestamp without time zone	arrival_t timestamp without time zone
1	1	2001-12-12 00:00:00	2001-12-12 00:00:00
2	2	2001-12-13 00:00:00	2001-12-15 00:00:00
3	17	2018-03-27 00:00:00	2019-09-19 00:00:00
4	36	2018-08-17 00:00:00	2019-10-21 00:00:00
5	41	2019-04-01 00:00:00	2019-03-03 00:00:00
6	69	2019-10-06 00:00:00	2019-04-19 00:00:00
7	136	2018-09-27 00:00:00	2019-10-15 00:00:00
8	151	2019-06-13 00:00:00	2018-12-03 00:00:00
9	156	2019-03-16 00:00:00	2018-06-10 00:00:00
10	159	2019-08-31 00:00:00	2018-04-02 00:00:00
11	216	2019-01-19 00:00:00	2018-06-22 00:00:00
12	234	2019-10-29 00:00:00	2018-05-08 00:00:00

users

	p_id [PK] integer	full_name text	pass_num integer
1	1	John Doe	1337
2	2	Jane Doe	21516516
3	11	6bc33c43aa	58
4	15	2862e8f84e	61
5	18	7d65eab3b2	77
6	37	a10606e5f9	19
7	63	f9a9f9a4bf	22
8	81	cfc39f8073	53
9	209	04c69db1a7	11
10	215	18ef7b1f88	50
11	292	0d3151280f	10
12	295	bfcb0140ac	35

Пункт 3

Пошук квитка за ID юзера(p_id) і типом квитка(type)



Пункт 4

Програмний код модуля “Model”

```
import random
import Connect
from View import View
import datetime
tables = {
    1: 'tickets',
    2: 'trains',
    3: 'users',
}

class Model:
    # Method that checks valid of the number of table that user input and returns it
    @staticmethod
    def validTable():
        incorrect = True
        while incorrect:
            table = input('Choose table number => ')
            if table.isdigit():
                table = int(table)
                if table >= 1 and table <= 3:
                    incorrect = False
                else:
                    print('Incorrect input, try again.')
            else:
```

```
            print('Incorrect input, try again.')
    return table

        # Method that prints all table of DB

@staticmethod
def showAllTables():
    connect = Connect.makeConnect()
    cursor = connect.cursor()
    for table in range(1, 4):
        table_name = """" + tables[table] + """"
        print(tables[table])

        show = 'select * from public.{}'.format(table_name)

        print("SQL query => ", show)
        print('')
        cursor.execute(show)
        records = cursor.fetchall()
        obj = View(table, records)
        obj.show()
    cursor.close()
    Connect.closeConnect(connect)

@staticmethod
def showOneTable():
    View.list()
    connect = Connect.makeConnect()
    cursor = connect.cursor()

    table = Model.validTable()

    table_name = """" + tables[table] + """"
    print(tables[table])

    show = 'select * from public.{}'.format(table_name)

    print("SQL query => ", show)
    print('')
    cursor.execute(show)
    records = cursor.fetchall()
    obj = View(table, records)
    obj.show()
    cursor.close()
```

```
    Connect.closeConnect(connect)

@staticmethod
def delete():
    connect = Connect.makeConnect()
    cursor = connect.cursor()
    restart = True
    while restart:
        View.list()
        table = Model.validTable()

        if table == 1:
            scname = input('Delete ticket with ID = ')
            delete = 'delete from "tickets" where "t_id"= {}'.format(scname)
            restart = False
        elif table == 2:
            clname = input('Delete train with ID = ')
            delete = 'delete from "trains" where "tr_id"=  {}'.format(clname)
            restart = False
        elif table == 3:
            dsname = input('Delete user with ID = ')
            delete = 'delete from "users" where "p_id"= {}'.format(dsname)
            restart = False
        else:
            print('\nIncorrect input, try again.')
    print(tables[table])
    print("SQL query => ", delete)
    cursor.execute(delete)
    connect.commit()
    print('Data deleted successfully!')
    cursor.close()
    Connect.closeConnect(connect)

@staticmethod
def insert():
    connect = Connect.makeConnect()
    cursor = connect.cursor()
    restart = True
    while restart:
        View.list()
        table = Model.validTable()
        if table == 1: #tickets
            t_id = input("Ticket ID = ")
            num = input("Car number = ")
```

```
            seat = input("Seat number = ")
            car_type = input('Car type = ')
            while 1:
                val = car_type
                if val == 'first' or val == 'second':
                    break
                else:
                    print("bad input")
                    car_type = input('Car type = ')

            p_id = input('Passanger ID = ')
            tr_id = input('Train ID = ')

            insert = 'insert into "tickets" ("t_id", "num", "seat", "type", "p_id", "tr_id") values ({}, {},
{}, \'{}\', {}, {})'.format(
                t_id, num, seat, car_type, p_id,tr_id)

            restart = False
        elif table == 2: #trains
            tr_id = input('Train ID = ')
            departure_t = input('Departure Time (YYYY-MM-DD HH:MM) = ')

            arrival_t = input('Arrival Time (YYYY-MM-DD HH:MM) = ')

            while 1:
                a = departure_t.split()[0].split('-')+departure_t.split()[1].split(':')
                b = datetime.datetime(int(a[0]),int(a[1]),int(a[2]),int(a[3]),int(a[4]))
                #print(b)
                c = arrival_t.split()[0].split('-')+arrival_t.split()[1].split(':')
                d = datetime.datetime(int(c[0]),int(c[1]),int(c[2]),int(c[3]),int(c[4]))

                #b = arrival_t.split()[0].split('-')+arrival_t.split()[1].split(':')
                #b = datetime.datetime(int(a[0]),int(a[1]),int(a[2]),int(a[3]),int(a[4]))

                if b<d:
                    break
                else:
                    print("bad input")
                    departure_t = input('Departure Time (YYYY-MM-DD HH:MM) = ')

                    arrival_t = input('Arrival Time (YYYY-MM-DD HH:MM) = ')

            insert = 'insert into "trains" ("tr_id", "departure_t", "arrival_t") values ({}, \'{}\',
\'{}\')'.format(
```

```
                tr_id, departure_t, arrival_t)

            restart = False
        elif table == 3: #users
            p_id = input('User ID = ')
            full_name = input('Full name = ')
            while 1:
                a = full_name.split(' ')
                if len(a) ==2 and (a[0].isalpha() and a[1].isalpha()):
                    a[0] = a[0].lower()
                    a[0] = a[0].capitalize()
                    a[1] = a[1].lower()
                    a[1] = a[1].capitalize()
                    #print(a)
                    full_name = a[0] + ' ' + a[1]
                    #print(full_name)
                    break
                else:
                    print('bad input')
                    full_name = input('Full name = ')
            pass_num = input('Passport number = ')



            insert = 'insert into "users" ("p_id", "full_name", "pass_num") values ({}, \'{}\',
{})'.format(p_id, full_name, pass_num)

            restart = False
        else:
            print('\nIncorrect input, try again.')
    print(tables[table])
    print('SQl query => ', insert)
    cursor.execute(insert)
    connect.commit()
    print('Data added successfully!')
    cursor.close()
    Connect.closeConnect(connect)

@staticmethod
def update():
    connect = Connect.makeConnect()
    cursor = connect.cursor()
    restart = True
    while restart:
```

```
View.list()
table = Model.validTable()
if table == 1:
    show = 'select * from public.tickets'
    cursor.execute(show)
    records = cursor.fetchall()
    obj = View(table, records)
    obj.show()
    #for row in obj.records:
    #    print(row[0])
    scname = "'" + input('Change row with t_id = ') + "'"

    View.attribute_list(1)

    in_restart = True
    while in_restart:
        num = input('Number of attribute =>')
        value = "'" + input('New value of attribute = ') + "'"
        if num == '1':
            set = '"t_id"= {}'.format(value)
            in_restart = False
        elif num == '2':
            set = '"num"= {}'.format(value)
            in_restart = False
        elif num == '3':
            set = '"seat"= {}'.format(value)
            in_restart = False
        elif num == '4':
            set = '"type"= {}'.format(value)
            in_restart = False
        elif num == '5':
            set = '"p_id"= {}'.format(value)
            in_restart = False
        elif num == '6':
            set = '"tr_id"= {}'.format(value)
            in_restart = False
        else:
            print('\nIncorrect input, try again.')
    update = 'update "tickets" set {} where "t_id"= {} returning t_id'.format(set, scname)
    restart = False
    pass
elif table == 2:
    show = 'select * from public.trains'
    cursor.execute(show)
```

```
        records = cursor.fetchall()
        obj = View(table, records)
        obj.show()
        #for row in obj.records:
        #    print(row[0])
        clname = "'" + input('Change row with tr_id = ') + "'"

        View.attribute_list(2)

        in_restart = True
        while in_restart:
            num = input('Number of attribute =>')
            value = "'" + input('New value of attribute = ') + "'"
            if num == '1':
                set = '"tr_id"= {}'.format(value)
                in_restart = False
            elif num == '2':
                set = '"departure_t"= {}'.format(value)
                in_restart = False
            elif num == '3':
                set = '"arrival_t"= {}'.format(value)
                in_restart = False
            else:
                print('\nIncorrect input, try again.')
        update = 'update "trains" set {} where "tr_id"= {} returning tr_id'.format(set, clname)
        restart = False
        pass
    elif table == 3:
        show = 'select * from public.users'
        cursor.execute(show)
        records = cursor.fetchall()
        obj = View(table, records)
        obj.show()
        #for row in obj.records:
        #    print(row[0])
        vroom = "'" + input('Change row with p_id = ') + "'"

        View.attribute_list(3)

        in_restart = True
        while in_restart:
            num = input('Number of attribute =>')
            value = "'" + input('New value of attribute = ') + "'"
            if num == '1':
```

```
                set = '"p_id"= {}'.format(value)
                in_restart = False
            elif num == '2':
                set = '"full_name"= {}'.format(value)
                in_restart = False
            elif num == '3':
                set = '"pass_num"= {}'.format(value)
                in_restart = False
            else:
                print('\nIncorrect input, try again.')
        update = 'update "users" set {} where "p_id"= {} returning p_id'.format(set, vroom)
        restart = False
        pass

    else:
        print('\nIncorrect input, try again.')
print(tables[table])
print("SQL query => ", update)


cursor.execute(update)
if cursor.rowcount == 0:
    print('No such id in the table, data unchanged')
else:
    connect.commit()
    print('Data updeted successfully!')
    cursor.close()
    Connect.closeConnect(connect)
    pass

@staticmethod
def random():
    connect = Connect.makeConnect()
    cursor = connect.cursor()

    incorrect = True
    while incorrect:
        num = input('How many users to random? => ')
        if num.isdigit():
            num = int(num)
            if num >= 1:
                incorrect = False
            else:
                print('Incorrect input, try again.')
```

```
        else:
            print('Incorrect input, try again.')

    insert = '''
    insert into "users" (p_id, full_name, pass_num)
    select (300*random())::integer+4,

    substr(md5(random()::text), 1, 10),
    (random() * 70 + 10)::integer
    FROM generate_series(1, {});

    '''.format(num)

    print("SQL query => ", insert)
    cursor.execute(insert)
    connect.commit()

    insert = '''
    insert into "trains" (tr_id, departure_t, arrival_t)
    select (300*random())::integer+4,

    DATE '2018-01-01' + (random() * 700)::integer,


    DATE '2018-01-01' + (random() * 700)::integer
    FROM generate_series(1, {});

    '''.format(num)

    print("SQL query => ", insert)
    cursor.execute(insert)
    connect.commit()

    print('Data randomed successfully!')
    cursor.close()
    Connect.closeConnect(connect)

@staticmethod
def text_search():
    connect = Connect.makeConnect()
    cursor = connect.cursor()
    restart = True
    while restart:
        incorrect = True
```

```
            while incorrect:
                mode = input("'
                1 -- mode 1
                2 -- test
                Choose mode = > "')
                if mode.isdigit():
                    mode = int(mode)
                    if mode >= 1 and mode <= 2:
                        incorrect = False
                    else:
                        print('Incorrect input, try again.')
                else:
                    print('Incorrect input, try again.')
            if mode == 1:
                print("Find by id and ticket type")
                inp = input("id > ")
                inp = int(inp)
                inp1 = input("type > ")
                restart = False
                text_search = "select * from users join tickets on (users.p_id=tickets.p_id) where
users.p_id={} and tickets.type = \'{}\'".format(inp, inp1)
            elif mode == 2:
                pass
            else:
                print('\nIncorrect input, try again.')

        #print(tables[table])
        print('SQL query => ', text_search)
        cursor.execute(text_search)
        #records = cursor.fetchall()
        #obj = View(table, records)
        #obj.show()
        rows = cursor.fetchall()
        print(rows)
        print('Data searched successfully!')
        cursor.close()
        Connect.closeConnect(connect)
```

Короткий опис функцій модуля

showAllTables() - виводить всі таблиці
showOneTable() -, виводить одну(обрану) таблицю
delete() - вилучає обраний запис з обраної таблиці
insert() - вставляє обраний запис в обрану таблиці
update() - редагує атрибут у вибраному рядку

random() - введення випадкових рядків у таблиці users та trains
search() - пошук квитка з заданим ідентифікатором користувача(p_id) і типу квитка(type